



ProbKB: Managing Web-Scale Knowledge

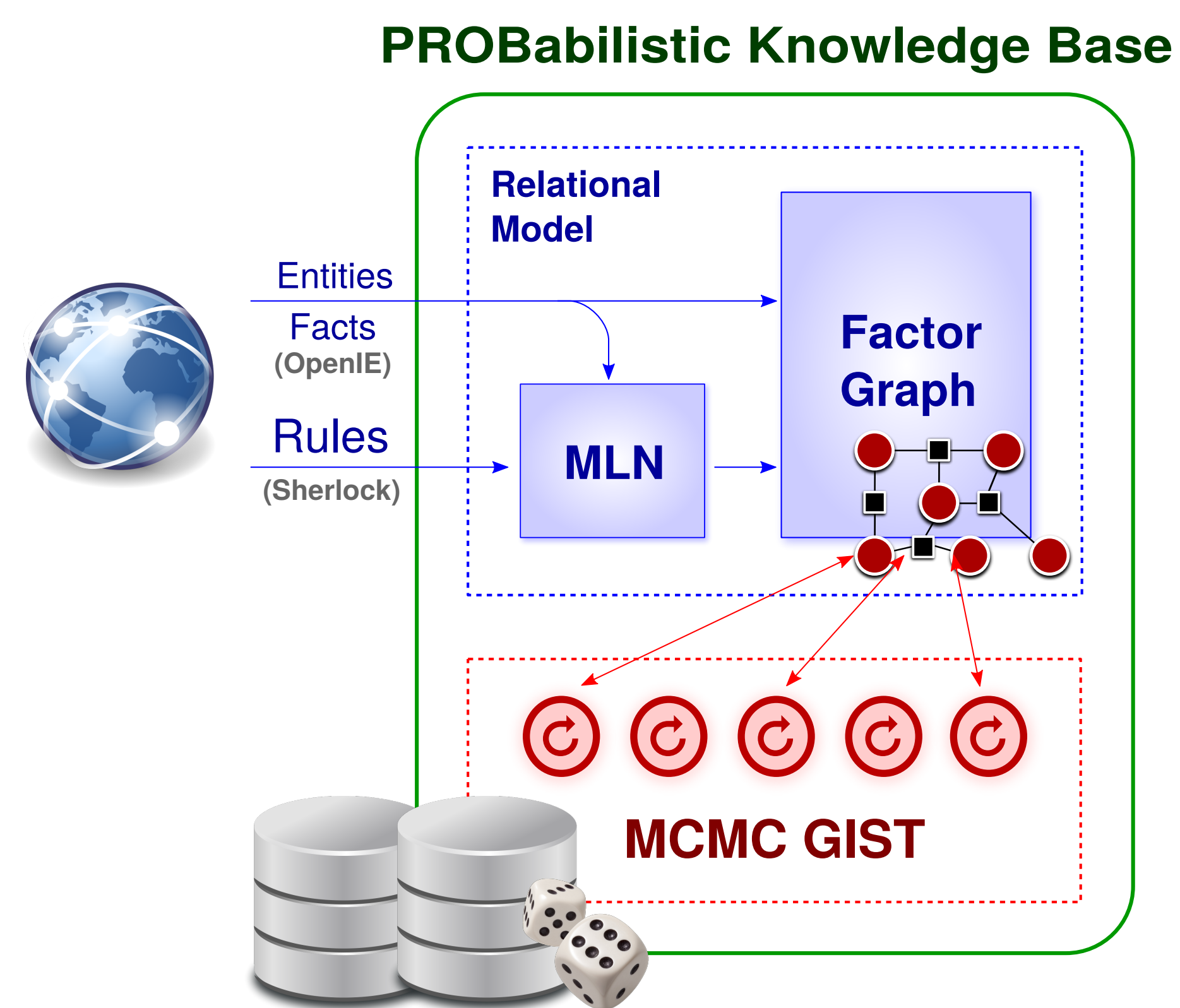
YANG CHEN[§], XING LIU[§]
[§]CISE, University of Florida
{yang,xinliu}@cise.ufl.edu



ABSTRACT

We present PROBKB, a PROBABILISTIC Knowledge Base constructed from web scale extracted entities, facts, and rules represented as a Markov logic network (MLN). We achieved web scale MLN inference by designing a novel structured, relational model for MLNs and efficient grounding algorithms that apply rules in batches. Errors are handled in a principled and elegant manner to avoid unnecessary resource consumption. The inference task is performed in Datapath, a data-centric parallel computation engine. Our initial experiments show that our approach has much better scalability than the state-of-the-art.

SYSTEM OVERVIEW



CHALLENGES AND CURRENT WORK

Knowledge Acquisition Markov logic networks.

Uncertainty Management Data cleaning, probabilistic graphical models, and sampling methods.

Scalability MPP frameworks and parallel computation engines.

RELATIONAL MLN MODEL

- We designed a relational model for MLN and pushed all extracted facts, and rules into the database. This allows grounding algorithms that apply rules in batches. We implemented this model on Greenplum, a massive parallel processing (MPP) framework.
- We identified six rules pattern in SHERLOCK dataset. Each rule type i has a table M_i recording the predicates involved in the rules of that type.
- We have another table R for relationships. For each relationship $p(x, y)$ that is stated in the text corpus, we have a tuple (p, x, y) in R .

p	q	r	p	x	y
p_1	q_1	r_1	p_1	x_1	y_1
p_2	q_2	r_2	p_1	x_2	y_2
p_3	q_3	r_3	p_2	x_1	y_1
p_4	q_4	r_4	p_2	x_2	y_2
...			...		

GROUNDING

- The relation model allows us to apply rules in batches using existing database techniques.
- Assume rules of type 3 are stored in table M3 (p, q, r), and relationships $p(x, y)$ are stored in $R(p, x, y)$, then the following SQL query computes atoms that are activated during the grounding process, this process is repeated until convergence, resulting in an active closure. The following SQL query then computes active clauses given the active atoms:

```
SELECT DISTINCT
R1.id AS id1, R2.id AS id2, R3.id AS id3
FROM M3 JOIN R R ON M3.p = R.p
JOIN R R1 ON M3.q = R1.p
JOIN R R2 ON M3.r = R2.p
WHERE R.x = R1.x AND R.y = R2.x AND R1.y = R2.y
```

- The result of grounding is a factor graph (Markov network). This graph encodes a probability distribution over its variable nodes, which can be used to answer user queries. We use TUFFY as comparison point. We tried to run PROBKB using the REVERB-SHERLOCK dataset and measured the grounding time in 85 seconds while TUFFY crashes before grounding.

KNOWLEDGE INTEGRITY

- We designed a novel robust semi-naive evaluation algorithm to improve accuracy and efficiency. The basic idea is to promote most confident facts and avoid repeated rule applications by maintaining a delta relation between two iterations.

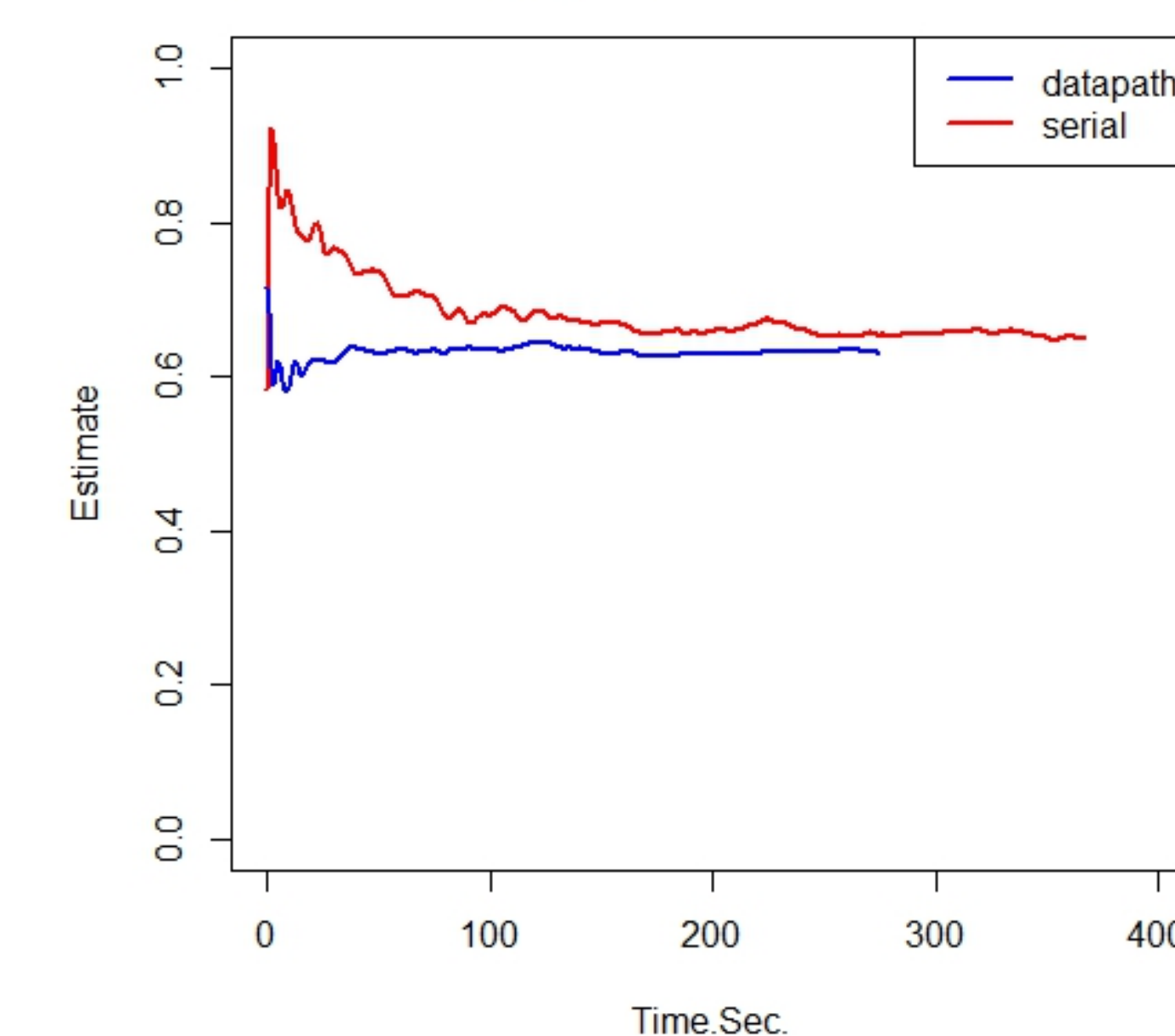
Algorithm 1 Robust Semi-Naive Evaluation

```
candidates ← all facts
beliefs ← promote (∅, candidates)
delta ← ∅
repeat
  upvotes ← infer (beliefs, delta)
  beliefs ← beliefs ∪ delta
  delta ← promote (upvotes, candidates)
until delta = ∅
```

INFERENCE

- Use MCMC algorithm to do the marginal inference in MLN. The results are the marginal probability of grounding facts.
- Our first approach is using Metropolis-Hastings algorithm with Datapath as our parallel engine. Partition the graph to subgraphs and runs the algorithm on each of them. We found this approach could break the Markov Logic Network if a factor is across several subgraphs.
- Our second approach is to use a write lock on each variable on the factor graph and random walk on the whole graph instead of partition the graph, and collecting samples in parallel.
- We sampled a subset with 700 facts and compared the time spent on generating 200 joint samples for PROBKB and TUFFY. PROBKB took 0.47 minute to complete the inference while TUFFY took 55 minutes.

50,000 vertices graph, parallel mcmc with write lock



50,000 vertices graph, parallel mcmc with write lock

